Frontend:
The frontend mainly provides a good user interface so that a client could have a good user experience on our website. For a good user interface, our front-end will make sure our webpage looks consistent. For example, the navigation bar will have a consistent background color, and the layout of buttons will look the same. To achieve consistency, frontend use the React framework; which will allow a frontend developer to make a large webpage into several small components. Making the web page into small components allows us to re-use some components again in the different webpage. Since we are re-using the components, we can make sure the consistency will be achieved. This type of design on Frontend provides a simple and consistent interface to an user. In the meantime, to reduce the time of letting the user see an information on a webpage, it is not a good idea to use server-side rendering. In other words, it is not good to ask a server to provide an entire webpage. If our frontend was using server-side rendering, users need to wait for the browser to retrieve a webpage on every time of accessing. However, our frontend is following single page principal through react and javascript. This means that our frontend will render a partial component in the webpage if the new data or component is required on the webpage. One single page design makes sure that the frontend will only re-render the necessary components instead of the entire webpage. In our current phase, we have implemented two main features on our frontend. Both of these features will interact with the backend through the API request. One example is profile page. Our frontend will use the API GET request to ask information of a person and then render these data into a personal profile page. If a user wants to change their information, our frontend will use an API POST request to the backend to ask the backend to update the data on the database.

Backend:
The backend is primarily in charge of providing and modifying relevant information at the behest of the frontend. As of currently, there are not many components composing the backend. The backend sets up a server in order to listen to the requests of the frontend. In the server, it sets up several listening posts, each in charge of a certain function (e.g. fetching profile, authenticating login, etc.). As of now, there aren't many features implemented, and so the backend has mostly been tasked with retrieving information from the database when called. It has to take information from the frontend and translate it to a form that can be understood by the database, and take information from the database and translate it to a form that can be understood by the frontend. For example, search_api.js takes care of search results from the search bar by searching the database, and also takes into account the filter options to further filter the results before sending the information to the frontend. This example also works in a try/catch function, just to catch any event in which talking to the database does not work.

System:
Our system focuses on the collaboration between the frontend, the backend and the database. For the database, we used neo4j to create a database. As long as our website is a social network website, we will need to deal with many relationships between different users. Thus, neo4j provides a convenient environment for creating relationships and nodes, which is the most suitable database management system for our website. For collaboration, the frontend will handle

users' actions and then send the request to the backend by using different APIs. When the backend receives the requests from the frontend, the backend will solve those requests by different API routers, these routers can access the database to query or update data. After the routers solve the request, they will send responses back to the frontend, and the frontend can display some expression to show the results of users' actions. In this system design, since all the work and responsibilities are well divided for laborers, we can efficiently manage different functions and components, and promote the performance of these functionalities.

Update (sprint 2):
We also plan to do the authentication checking before any handle any api requests, so we can make sure that the request is from legal user.

CRC Cards:

<table>
<tr><td colspan="2">Class Name:

- search_bar.js</td></tr>
<tr><td colspan="2">Parent Class:
- Main.js
Subclass:</td></tr>
<tr><td>Responsibilities:
- Display search bar and filters
- Handle search
- Handle filters</td><td>Collaborators:
- api.js
- App.css</td></tr>
</table>

<table>
<tr><td colspan="2">Class Name:

- ProfileForm.js</td></tr>
<tr><td colspan="2">Parent Class:
- Main.js
Subclass:</td></tr>
<tr><td>Responsibilities:
- Show a user's information
- Allow modification of user's information</td><td>Collaborators:
- api.js</td></tr>
</table>

Class Name: - top_bar.js	
Parent Class: - Main.js Subclass: - UserTools.js	
Responsibilities: - Display home, feature, pricing and FAQs - Handle switch between pages - Show notifications	Collaborators: - api.js - App.css

Class Name: - profile_fetch_api.js	
Parent Class: - index.js Subclass:	
Responsibilities: - Fetch user's info from database - Send fetched info to frontend	Collaborators: - ProfileForm.js

Class Name: - PageContext.js	
Parent Class: - Main.js Subclass:	
Responsibilities: - Manage different pages - Switch the pages that display	Collaborators:

Class Name: - login.js	
Parent Class: - App.js Subclass:	
Responsibilities: - Allows users to login - Check login info for correctness - Tell App results of correctness check	Collaborators: - api.js - signin_api.js - cookie_api.js

Class Name: - api.js	
Parent Class: - index.js Subclass:	
Responsibilities: - Manage different api - Allow other components to call api - Return response to other components	Collaborators:

Class Name: - neo4j.js	
Parent Class: - index.js Subclass:	
Responsibilities: - Setup neo4j connection - Allow other components in the backend to create a neo4j session	Collaborators:

Class Name: - index.js (backend)	
Parent Class: Subclass: - Every component in the backend	
Responsibilities: - Setup server - Listen to connections and requests	Collaborators:

Class Name: - index.js (frontend)	
Parent Class: Subclass: - Every component in the frontend	
Responsibilities: - Setup ReactDOM - Setup WebVitals - Be the basis of the webpage	Collaborators:

Class Name: - App.js	
Parent Class: - index.js Subclass: - Main.js - login.js	
Responsibilities: - Check authentication - Collaborate with PageContext.js to manage different webpages	Collaborators: - api.js - PageContext.js

Class Name: - CalendarPage.js	
Parent Class: - Main.js Subclass: - Calendar.jsx - CreateEventForm.jsx	
Responsibilities: - Control what kind of form should be show on the calendar page based on user's action	Collaborators:

Class Name: - calendar_api.js (back-end)	
Parent Class: - index.js (back-end)	
Responsibilities: - Fetch all user's calendar info - Store newly created entries - Update existing entries - Delete existing entries	Collaborators: - api.js

Class Name: - Calendar.jsx	
Parent Class: - CalendarPage.js Subclass:	
Responsibilities: - Show user's event on Calendar	Collaborators: - api.js

Class Name: - CreateEventForm.jsx	
Parent Class: - CalendarPage.js Subclass:	
Responsibilities: - Let user to fill some details of future event	Collaborators: - Api.js

Class Name: - UpdateEventForm.jsx	
Parent Class: - CalendarPage.js Subclass:	
Responsibilities: - Let user to update existence event	Collaborators: - api.js

Class Name: - FriendList.js	
Parent Class: - Main.js Subclass:	
Responsibilities: - Allow user to modify the friend list, they can remove their friends, unfollow the user, followback their followers - Allow users to view the friend requests from other users, and allow users to accept or reject - Allow user to search for the users in the friendList - Allow user to view other users' information	Collaborators: - topbar.js - Topbar.js - PageContext.js - UserPage.js

Class Name: - Home.js	
Parent Class: - Main.js Subclass:	
Responsibilities: - Display newest post - Display the daily calendar	Collaborators: - api.js - PostCard.js - DailyCalendarCard.js

Class Name: - PostCard.js	
Parent Class: Subclass:	
Responsibilities: - Display post titles, comments and likes	Collaborators: - api.js - PostPage.js - Home.js

Class Name: - PostsPage.js	
Parent Class: - Main.js Subclass: - PostCard.js	
Responsibilities: - Display all the posts	Collaborators: - api.js - App.css - Home.js

Class Name: - UserPage.js	
Parent Class: - Main.js Subclass:	
Responsibilities: - Display user's information - Allow other users to send friend requests or follow	Collaborators: - api.js - FriendList.js - PageContext.js

Class Name: - UserTools.js	
Parent Class: - TopBar.js Subclass:	
Responsibilities: - Allow users to open profile, settings, and logout	Collaborators: - PageContext.js